

# Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems

W. Matthew Carlyle  
Johannes O. Royset  
R. Kevin Wood

Operations Research Department  
Naval Postgraduate School  
Monterey, California, USA  
8 July 2005

---

## Abstract

We present a new algorithm for solving the *constrained shortest-path problem* (CSPP), which generalizes the standard shortest-path problem by adding one or more path-weight side constraints. Our procedure Lagrangianizes those constraints, optimizes the resulting Lagrangian function and then closes any duality gap by enumerating near-shortest paths, measured with respect to the Lagrangianized length. “Near-shortest” implies  $\epsilon$ -optimal, with  $\epsilon$  equaling the duality gap. Our recently developed algorithm for enumerating near-shortest paths leads to an algorithm for CSPP that proves, empirically, to be significantly faster than the fastest algorithm documented in the literature. We solve problems with millions of vertices and edges and up to ten side constraints, in few seconds on a laptop computer.

---

## 1 Introduction

Shortest-path problems in networks with non-negative edge lengths (or with some negative-length edges, but no negative-length cycles) can be solved efficiently, both in theory and practice (e.g. Ahuja et al. [1], pp. 93-157). However, if each edge possesses a non-negative weight in addition to its length, and if a single side constraint is placed on the optimal path’s total weight, the problem becomes NP-complete (Garey and Johnson [14], p. 214). Multiple edge weights and weight limits may be defined, and the general problem is known as the *constrained shortest-path problem* (CSPP). This paper develops an improved algorithm and ancillary techniques for solving this problem.

CSPP is NP-complete in the weak sense for a fixed number of side constraints and admits a dynamic-programming solution procedure (Joksch [17]). However, dynamic programming (DP) can be unacceptably slow in practice even with a single side constraint, and label-setting algorithms based on DP have supplanted straightforward DP implementations (e.g., Aneja et al. [2], Dumitrescu and Boland [10]). Other potentially useful techniques include branch and bound using a Lagrangian-based bound (Beasley and Christofides [3]), and Lagrangian relaxation coupled with  $K$ -shortest-paths enumeration (Handler and Zang [16]).

Dumitrescu and Boland [10] describe a label-setting algorithm, combined with several pre-processing techniques, that may be the most efficient technique currently available for CSPP. We present a competitive approach, which we call *Lagrangian relaxation with near-shortest paths enumeration* (LRE). This approach Lagrangianizes the side constraints, optimizes the Lagrangian function, and closes any duality gap by enumerating near-shortest paths. Path length is measured with respect to the Lagrangianized edge lengths, and “near shortest” means  $\epsilon$ -optimal, with  $\epsilon$  equaling the duality gap. Often, a feasible solution is identified while optimizing the Lagrangian function, but any upper bound on the optimal objective value will suffice, in theory, for the method to succeed.

LRE resembles the algorithm of Handler and Zang, but with a near-shortest-paths (NSP) algorithm replacing their  $K$ -shortest-paths (KSP) algorithm. The LRE approach seems particularly attractive because we (Carlyle and Wood [6]) have recently developed an extremely fast near-shortest-paths algorithm, and “near-shortest paths” proves to be a more appropriate paradigm than “ $K$ -shortest paths” for the enumerative phase of the algorithm. We discuss this issue in more detail later. LRE is also similar to the branch-and-bound algorithm of Beasley and Christofides, but LRE does not solve a shortest-path problem at each node of the branch-and-bound enumeration tree.

CSPP arises in a number of real-world contexts. The most well-known application may be column-generation for generalized set-partitioning models of crew-scheduling and crew-rostering problems, especially in the airline industry (e.g., Gamache et al. [13], Day and Ryan [7], Vance et al. [25]). Other important applications include minimum-risk routing of

military vehicles and aircraft (e.g., Boerman [4], Latourell, et al. [20], Lee [21], Zabaranin et al. [28]), signal routing in communications networks having quality-of-service guarantees (see Korkmaz and Krunz [19] and the references therein), signal compression (Nygaard et al. [23]) and numerous transportation problems (e.g., Nachtigall [22], Kaufman and Smith [18]).

The remainder of the paper begins by defining CSPP precisely, and by then describing the basic LRE solution approach. Next, we provide an overview of the NSP algorithm that makes our LRE algorithm viable. (An appendix contains detailed pseudo-code.) We do not discuss optimizing the Lagrangian function in any detail because the relevant techniques are well known. We then refine the basic LRE algorithm by adding a simple pre-processing scheme and aggregated-bounds tests to limit enumeration in the NSP algorithm. We also describe a technique for identifying feasible solutions in especially tightly constrained problems. Finally, we present computational results for the LRE algorithm applied to small-, medium-, and large-scale networks. The structure of some of these problems mimics that of the problems solved by Dumitrescu and Boland [10], and therefore allows for useful comparisons. We also present computational examples that derive from a model of minimum-risk routing for a military convoy.

## 2 Problem Definition and Basic Approach

Let  $G = (V, E)$  be a *directed graph* with vertex set  $V$  and edge set  $E$ . Each *edge*  $e = (u, v) \in E$  connects distinct *vertices*  $u, v \in V$ . It possesses *length*  $c_e \geq 0$  and one or more *weights*,  $f_{ie} \geq 0$ , for  $i \in I$ , where  $I$  indexes a set of *side constraints*. Each side constraint  $i$  has a *weight limit*  $g_i \geq 0$  defined.

A *directed s-t path*  $E_P$  is an ordered set of edges of the form  $E_P = \{(s, v_1), (v_1, v_2), \dots, (v_{k-1}, t)\}$ . The path is *simple* if no vertices are repeated. Given two distinct vertices  $s, t \in V$ , the *constrained shortest-path problem* (CSPP) seeks a directed, simple, *s-t path*  $E_P$  such that  $\sum_{e \in E_P} f_{ie} \leq g_i$  for all  $i \in I$ , and such that  $\sum_{e \in E_P} c_e$  is minimized.

Let  $A$  denote the  $|V| \times |E|$  vertex-edge incidence matrix for  $G$  such that if  $e = (u, v)$ ,

then  $A_{ue} = 1$ ,  $A_{ve} = -1$  and  $A_{we} = 0$  for any  $w \neq u, v$ . Also, let  $b_s = 1$ ,  $b_t = -1$  and  $b_v = 0$  for all  $v \in V \setminus \{s, t\}$ , and let  $\mathbf{g}$  denote the vector  $(g_1 g_2 \cdots g_{|I|})$ . For each  $i \in I$ , we collect the edge weights  $f_{ie}, e \in E$ , in the row vector  $\mathbf{f}_i$ . Finally, we define  $F$  as the  $|I| \times |E|$ -matrix having vectors  $\mathbf{f}_i$  as its rows. Then, CSPP may be written as this integer program (Ahuja et al. [1], p. 599),

$$\text{CSPIP} \quad z^* \equiv \min_{\mathbf{x}} \mathbf{c}\mathbf{x} \quad (1)$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \quad (2)$$

$$F\mathbf{x} \leq \mathbf{g} \quad (3)$$

$$\mathbf{x} \geq \mathbf{0} \text{ and integer,} \quad (4)$$

where  $x_e^* = 1$  if edge  $e$  is in the optimal path, and  $x_e^* = 0$ , otherwise. Note that the problem's structure leads to binary solutions without explicit constraints  $\mathbf{x} \leq \mathbf{1}$ . Equations (3) are CSPP's side constraints. We refer to  $\hat{\mathbf{x}}$  as a “path” when it satisfies all constraints of **CSPIP** except possibly the side constraints. Strictly speaking, such an  $\hat{\mathbf{x}}$  could have  $\hat{x}_e = 1$  for all edges  $e$  around one or more cycles, but (i) there always exists an optimal solution without cycles since  $\mathbf{c} \geq \mathbf{0}$  is assumed, and (ii) our LRE algorithm cannot generate any solutions to CSPP that have cycles in them, so (iii) this point can be safely ignored.

**CSPIP** would define an easy-to-solve shortest-path problem if not for the side constraints. We expect to have only a few such constraints, say one to ten, and it therefore seems reasonable to base a solution procedure on relaxing them. Using the standard theory of Lagrangian relaxation (e.g., Ahuja et al. [1], pp. 598-648), we know that for any appropriately dimensioned row vector  $\boldsymbol{\lambda} \geq \mathbf{0}$ ,

$$z^* \geq \underline{z}(\boldsymbol{\lambda}) \equiv \min_{\mathbf{x}} \mathbf{c}\mathbf{x} + \boldsymbol{\lambda}(F\mathbf{x} - \mathbf{g}) \quad (5)$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \quad (6)$$

$$\mathbf{x} \geq \mathbf{0} \text{ and integer.} \quad (7)$$

We then rewrite the objective function, and optimize the Lagrangian lower bound  $\underline{z}(\boldsymbol{\lambda})$

through

$$\mathbf{CSPLR} \quad \underline{z}^* \equiv \max_{\boldsymbol{\lambda} \geq \mathbf{0}} \underline{z}(\boldsymbol{\lambda}) \quad (8)$$

$$= \max_{\boldsymbol{\lambda} \geq \mathbf{0}} \min_{\mathbf{x}} (\mathbf{c} + \boldsymbol{\lambda}F)\mathbf{x} - \boldsymbol{\lambda}\mathbf{g} \quad (9)$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \quad (10)$$

$$\mathbf{x} \geq \mathbf{0} \text{ and integer.} \quad (11)$$

For any fixed  $\boldsymbol{\lambda} \geq \mathbf{0}$ , computing  $\underline{z}(\boldsymbol{\lambda})$  simply requires the solution of a shortest-path problem with Lagrangian-modified edge lengths.

The solutions of the inner minimization of **CSPLR** are intrinsically integer, so we know that  $\underline{z}^*$  equals the optimal objective value of the linear-programming relaxation of **CSPIP** (e.g., Fisher [11]). And, it is easy to construct examples in which this bound is not very close to  $z^*$ . Thus, the success of the LRE approach will sometimes depend on the ability to close a large duality gap quickly.

The outer maximization over  $\boldsymbol{\lambda}$  can be solved in several ways, depending on the dimension of  $\boldsymbol{\lambda}$ , i.e., on the number of side constraints. Bisection search works well for a single side constraint (Fox and Landi [12]), as does coordinate search for a few side constraints (e.g., DeWolfe et al. [8]). Beasley and Christofides [3] describe the use of subgradient optimization for CSPP with up to 10 side constraints. A constraint-generation algorithm analogous to Benders decomposition could also be used (e.g., Wolsey [27], pp. 172-173)

In the process of optimizing  $\underline{z}(\boldsymbol{\lambda})$ , we may find a solution  $\hat{\mathbf{x}}$  that is feasible with respect to the relaxed side constraints (3) as well as to constraints (2) and (4). Such a feasible solution provides an upper bound  $\bar{z} = \mathbf{c}\hat{\mathbf{x}} \geq z^*$ . In particular, if **CSPIP** possesses only a single side constraint, then for sufficiently large  $\boldsymbol{\lambda}$  every optimal solution of **CSPLR** satisfies (3). Unfortunately, as the number of side constraints grows, finding a feasible solution to **CSPIP** during the optimization of  $\underline{z}(\boldsymbol{\lambda})$  becomes less and less likely. Fortunately, since every optimal solution of **CSPIP** will be a simple path, a (weak) upper bound for a feasible **CSPIP** is always  $\bar{z} = (|V| - 1)c_{\max}$  where  $c_{\max} \equiv \max_{e \in E} c_e$ .

Now, given  $\bar{z}$ , and an optimal (or nearly optimal) Lagrangian vector  $\boldsymbol{\lambda}$ , the following

theorem and corollary show that we may view the problem of identifying  $\mathbf{x}^*$ , an optimal solution to **CSPIP**, as one of simple enumeration. (The theorem is implicit in Handler and Zang [16].)

**Theorem 1** *Let  $\hat{X}(\boldsymbol{\lambda}, \bar{z})$  denote the set of feasible solutions  $\hat{\mathbf{x}}$  to **CSPLR** with the property that  $\mathbf{c}\hat{\mathbf{x}} + \boldsymbol{\lambda}(F\hat{\mathbf{x}} - \mathbf{g}) \leq \bar{z}$ . Then,  $\mathbf{x}^* \in \hat{X}(\boldsymbol{\lambda}, \bar{z})$ .*

*Proof:* Since  $F\mathbf{x}^* \leq \mathbf{g}$  and  $\boldsymbol{\lambda} \geq \mathbf{0}$ , the result follows from the facts that (i)  $\mathbf{c}\mathbf{x}^* + \boldsymbol{\lambda}(F\mathbf{x}^* - \mathbf{g}) \leq z^*$ , and (ii)  $z^* \leq \bar{z}$ . ■

**Corollary 1** *If **CSPIP** is feasible, an optimal solution  $\mathbf{x}^*$  can be identified by enumerating  $\hat{X}(\boldsymbol{\lambda}, \bar{z})$  and selecting*

$$\mathbf{x}^* \in \underset{\mathbf{x} \in \hat{X}(\boldsymbol{\lambda}, \bar{z})}{\operatorname{argmin}} \{ \mathbf{c}\mathbf{x} \mid F\mathbf{x} \leq \mathbf{g} \}. \quad (12)$$

■

Theorem 1 and Corollary 1 are valid for any  $\boldsymbol{\lambda} \geq \mathbf{0}$ , but it is easy to devise examples that show how an optimal or near-optimal  $\boldsymbol{\lambda}$  for **CSPLR** can exponentially reduce the size of  $\hat{X}(\boldsymbol{\lambda}, \bar{z})$ , and thereby exponentially reduce computational workload. (Naturally, improving the upper bound  $\bar{z}$  can also reduce the workload exponentially.)

Theorem 1 and Corollary 1 imply that we may need to enumerate each path  $\hat{\mathbf{x}}$  satisfying  $(\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \leq \bar{z}$ . That is, if  $\mathbf{x}_\lambda^*$  solves the shortest-path problem given the edge-length vector  $\mathbf{c} + \boldsymbol{\lambda}F$ , and  $\underline{z}(\boldsymbol{\lambda}) = (\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}}_\lambda^* - \boldsymbol{\lambda}\mathbf{g}$  then CSPP is solved by enumerating all paths  $\hat{\mathbf{x}}$  such that  $\underline{z}(\boldsymbol{\lambda}) \leq (\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \leq \bar{z}$ . In turn, this means that, given edge-length vector  $\mathbf{c} + \boldsymbol{\lambda}F$ , and including the Lagrangian constant term  $-\boldsymbol{\lambda}\mathbf{g}$  in the length of any path, we wish to find all  $\epsilon$ -optimal (near-shortest) paths for  $\epsilon \equiv \bar{z} - \underline{z}(\boldsymbol{\lambda})$ . Naturally, as any enumeration process proceeds, better feasible solutions may be found,  $\bar{z}$  and thus  $\epsilon$  may consequently improve, and the enumeration may consequently be reduced.

From the above discussion, it appears that an NSP algorithm, which identifies  $\epsilon$ -optimal paths, is a natural choice for path enumeration in the LRE solution approach to CSPP. A typical  $K$ -shortest-paths (KSP) algorithm could be used, however (e.g., Hadjiconstantinou

and Christofides [15]), Such an algorithm is meant to enumerate the  $K$  shortest paths in a network for a pre-specified integer  $K$ . But, because it enumerates paths in order of increasing length, it could be halted when path length exceeds  $(\mathbf{c} + \boldsymbol{\lambda}F)\mathbf{x}_{\boldsymbol{\lambda}}^* + \epsilon$ . However, enumerating paths in order of length requires unnecessary computational work, storage and algorithmic complexity. The NSP algorithm of [6] is much simpler and faster. “Faster” is a valid claim here, even though NSP and KSP are somewhat different problems. Wood and Carlyle [6] show that solving KSP using an NSP algorithm as a subroutine is the fastest available method by a wide margin (assuming the paths need not be ordered by length).

### 3 The LRE Algorithm for CSPP

This section outlines the basic LRE algorithm for CSPP.

#### LRE Algorithm for CSPP (Outline)

1. Find  $\boldsymbol{\lambda}$  that optimizes, or approximately optimizes, the Lagrangian lower bound  $\underline{z}(\boldsymbol{\lambda})$ .
2. Let  $\hat{X}$  denote the set of feasible paths identified while optimizing  $\underline{z}(\boldsymbol{\lambda})$ . If  $\hat{X} \neq \emptyset$ , set upper bound  $\bar{z} = \min_{\hat{\mathbf{x}} \in \hat{X}} \mathbf{c}\hat{\mathbf{x}}$ , else set  $\bar{z} = (|V| - 1)c_{\max} + \gamma$  for some  $\gamma > 0$ .
3. Using a standard path-enumeration procedure (e.g., Byers and Waterman [5]), begin enumerating all paths  $\hat{\mathbf{x}}$  such that  $(\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \leq \bar{z}$ , with the following modifications:
  - (a) Use  $\bar{z}$  and the side constraints to limit the enumeration when it can be projected that the current path cannot be extended to one whose length improves upon  $\bar{z}$  or which does not violate the side constraints.
  - (b) Whenever a feasible path,  $\hat{\mathbf{x}}$ , is found that is shorter than the incumbent, update the incumbent to  $\hat{\mathbf{x}}$  and update the upper bound to  $\bar{z} = \mathbf{c}\hat{\mathbf{x}}$ .
4. If no  $\hat{\mathbf{x}}$  is found in Step 3, the problem is infeasible; otherwise the best solution  $\hat{\mathbf{x}}$  is optimal.

The NSP algorithm upon which we base this procedure (see the Appendix for the complete pseudo-code) begins by

1. Computing the minimum “Lagrangian distance”  $d(v)$  from each  $v \in V$  back to  $t$  by solving a single, backwards, shortest-path problem starting from  $t$ , using Lagrangianized edge lengths  $\mathbf{c}' \equiv \mathbf{c} + \boldsymbol{\lambda}F$ ,
2. Computing analogous minimum  $v$ -to- $t$  distances  $d_0(v)$  for all  $v \in V$  with respect to edge lengths  $\mathbf{c}$ , and
3. Computing analogous minimum  $v$ -to- $t$  distances  $d_i(v)$  for all  $v \in V$  and  $i \in I$  with respect to edge weights  $\mathbf{f}_i$ .

This first phase requires the solution of only  $|I| + 2$  shortest-path problems.

Let  $E_P(u) = \{(s, v_1), (v_1, v_2), \dots, (v_{k-1}, u)\}$  denote a *directed  $s$ - $u$  subpath*. In the second phase of the algorithm, a standard path-enumeration algorithm commences from  $s$ , but extends subpath  $E_P(u)$  along edge  $e = (u, v)$  if and only if the following conditions hold:

1.  $E_P(u) \cup \{e\}$  can be extended to a path whose Lagrangianized length does not exceed  $\bar{z}$ , i.e.,  $L(u) + (c_e + \sum_{i \in I} \lambda_i f_{ie}) + d(v) \leq \bar{z}$ , where  $L(u)$  denotes the Lagrangianized length of  $E_P(u)$  and where, by convention, we define  $L(s) = -\boldsymbol{\lambda}g$ .
2.  $E_P(u) \cup \{e\}$  can be extended to a path whose true length is strictly less than  $\bar{z}$ , i.e.,  $L_0(u) + c_e + d_0(v) < \bar{z}$ , where  $L_0(u)$  denotes the length of  $E_P(u)$ .
3. For all  $i \in I$ ,  $E_P(u) \cup \{e\}$  can be extended to a path whose  $i$ -th weight does not exceed  $g_i$ , i.e.,  $L_i(u) + f_{ie} + d_i(v) \leq g_i$ , where  $L_i(u)$  denotes the  $i$ -th total weight of  $E_P(u)$ .
4. The path does not loop back on itself.

It is easy to see that these conditions are necessary for existence of a feasible path, because the labels  $d(v)$ ,  $d_0(v)$  and  $d_i(v)$  represent lower bounds on the the Lagrangianized length, true length, or  $i$ -th weight, respectively, that are required to complete the subpath  $E_P(u) \cup \{e\}$ . (Computer scientists will recognize this algorithm as a non-heuristic version of “A\* search,”

e.g., Russell and Norvig 1995, pp. 92-107.) These labels are lower bounds, not exact values, because the  $v$ - $t$  paths they represent may use vertices in the current subpath  $E_P(u)$ . Such vertices cannot be used together with the current subpath because their inclusion would create at least one illegal cycle in the final solution.

Exact values can be maintained for those labels if we recompute the various “distances” every time we extend or retract the current subpath, and ensure that those computations avoid traversing vertices on that subpath. Exact labels would make the tests stronger and could reduce enumeration substantially. Indeed, when enumerating near-shortest paths with respect to a single distance measure, this recomputation ensures that only polynomial work need be expended for each path enumerated; lacking these recomputations, the work can be exponential (Carlyle and Wood [6]).

On the other hand, solving the shortest-path problems required to maintain precise distance labels can add tremendously to the computational workload. This workload need not be as great as one shortest-path calculation for each type of distance label, for every extension or retraction of the current subpath, but the workload can still be onerous. Indeed, Carlyle and Wood show empirically that, for enumerating near-shortest paths, an algorithm that does not recompute distances can be orders of magnitude faster than one that does. This holds true over a wide range of problem classes, even though the worst-case complexity worsens. Consequently, we do not maintain precise distance labels in our LRE algorithm.

(Note: Even if we recomputed distance labels, the amount of work per feasible path enumerated in LRE could be exponential. This is true because tests 1-3 above, even with precise distance labels, are necessary, but not sufficient to guarantee the existence of a single path that satisfies all the conditions simultaneously.)

The LRE algorithm actually defines a branch-and-bound procedure that incorporates a depth-first enumeration tree along with feasibility checks. Branching consists of extending the current subpath by one edge. A linear-programming-based algorithm would update the lower-bounding model to account for the restriction imposed by a branch and would then reoptimize the lower bound. LRE updates the bound, but does not reoptimize it.

Reoptimization would require a new search over  $\lambda$  and the solution of more shortest-path problems, which we have not found to be computationally effective.

As with any branch-and-bound procedure, allowing a small but acceptable optimality gap in LRE can substantially reduce the amount of enumeration required. The pseudo-code for the NSP algorithm, given in the Appendix, does include an “absolute-gap parameter” for this purpose,  $\delta \geq 0$ .

## 4 Algorithmic Enhancements

As demonstrated in the next section, the basic LRE algorithm can solve a variety of problems quickly. This section describes, however, three enhancements to the basic algorithm which prove useful for solving more difficult problems.

### 4.1 Pre-processing

A pre-processing procedure may be able to identify numerous vertices and edges that cannot lie on any optimal path, and remove them prior to optimization. Such pre-processing can reduce subsequent computation time by reducing the size of the network on which the rest of the algorithm must operate. Furthermore, a smaller network may also yield a tighter Lagrangian bound. We use the following pre-processing procedure originally proposed by Aneja et al. [2]:

1. For all  $i \in I$  and  $v \in V$ , compute a minimum-weight  $s$ - $v$  subpath length  $D_i(v)$  and a minimum-weight  $v$ - $t$  subpath length  $d_i(v)$  using the  $i$ -th weight.
2. Delete any edge  $e = (u, v) \in E$  such that  $D_i(u) + f_{ie} + d_i(v) > g_i$  for any  $i \in I$ .
3. Repeat steps 1 and 2 until no new edges can be deleted.

Once an upper bound  $\bar{z}$  is found, this procedure can also be performed with respect to the edge lengths  $\mathbf{c}$  and Lagrangianized edge lengths  $\mathbf{c} + \lambda F$ , for any  $\lambda$ , with “ $> g_i$ ” replaced by “ $\geq \bar{z}$ .” (Recall that we include the Lagrangian constant term  $-\lambda \mathbf{g}$  in the Lagrangian path length.)

A similar procedure for eliminating vertices can also be constructed (for example, see Aneja et al. [2] and Dumitrescu and Boland [10]), but the edge-elimination procedure essentially subsumes it.

By its construction, our NSP algorithm automatically performs many of the checks that a pre-processing procedure carries out. However, we do find that computational improvements can accrue when we perform pre-processing with respect to the weights as in Step 1-3 above. Since the subpaths in Step 1 are computed for each weight individually, there may be no feasible  $s$ - $t$  path involving an edge even when the edge was not deleted. To increase the number of edges removed, we also pre-process with respect to the aggregated weight  $\sum_{i \in I} f_{ie}$ , where “ $> g_i$ ” is replaced by “ $> \sum_{i \in I} g_i$ ” in Step 2. This “aggregated test” considers all the weights for each edge along subpaths simultaneously and hence has the potential to remove additional edges as the following example illustrates: Consider a three-vertex network with edges  $a = (s, 2)$ ,  $b = (2, t)$ , and  $c = (2, t)$ , two weights  $f_{1a} = f_{2a} = f_{1b} = f_{2c} = 1$ ,  $f_{1c} = f_{2b} = 2$ , and weight limits  $g_1 = g_2 = 2$ . Trivially, edge  $a$  cannot be removed by checks on the individual weights since there are  $2$ - $t$  subpaths with weight 1 for both weights. However, there are no  $2$ - $t$  subpaths with aggregated weight of 2 and hence edge  $a$  is deleted using the aggregated test. “LRE-P” will denote the LRE algorithm that incorporates pre-processing Step 1-3 with respect to individual weights as well as the aggregated weight.

## 4.2 Aggregated Bounds to Limit Enumeration

Once  $\lambda$  has been optimized, the path-enumeration portion of the LRE algorithm continually asks: Given that  $x_e$  must equal 1 if  $e = (u, v)$  is on the current subpath, can this subpath be extended to a complete path  $\mathbf{x}$  such that

$$\begin{array}{ll} \text{[A]} & \mathbf{c}\mathbf{x} < \bar{z}, \quad \text{and} \\ \text{[B]} & -\lambda\mathbf{g} + (\mathbf{c} + \lambda F)\mathbf{x} \leq \bar{z}, \quad \text{and} \\ \text{[C]} & F\mathbf{x} \leq \mathbf{g}? \end{array}$$

We do not extend the path along an edge, say  $e = (v, w)$ , if setting  $x_e = 1$  would force  $\mathbf{x}$  to violate any of the constraints [A], [B], and [C]. The constraints are checked independently, however, so this does not guarantee that a feasible completion (with respect to [A] and [B]

and  $[C]$ ) of the current subpath exists.

Similar to pre-processing with respect to an aggregated weight described in the previous sub-section, we consider aggregated version of  $[A]$ ,  $[B]$ , and  $[C]$  that may further limit the path enumeration. Using empirically determined scaling factors, we construct tests that aggregate  $[C]$ , each pair of  $[A]$ ,  $[B]$ , and  $[C]$ , and one that aggregates all three:

$$\mathbf{g}^{-1}[C] \tag{13}$$

$$[A] + [B] \tag{14}$$

$$\underline{z}^{-1}[A] + \mathbf{g}^{-1}[C] \tag{15}$$

$$\underline{z}^{-1}[B] + \mathbf{g}^{-1}[C] \tag{16}$$

$$\underline{z}^{-1}[A] + \underline{z}^{-1}[B] + \mathbf{g}^{-1}[C] \tag{17}$$

where  $\mathbf{g}^{-1} = (1/g_1 \dots 1/g_{|I|})$  and  $\underline{z}^{-1} = 1/\underline{z}(\boldsymbol{\lambda})$ . For instance, checking  $\underline{z}^{-1}[B] + \mathbf{g}^{-1}[C]$  corresponds to testing whether or not

$$-\underline{z}^{-1}\boldsymbol{\lambda}\mathbf{g} + (\underline{z}^{-1}\mathbf{c} + \underline{z}^{-1}\boldsymbol{\lambda}F + \mathbf{g}^{-1}F)\mathbf{x} < \underline{z}^{-1}\bar{z} + |I| \tag{18}$$

All of these tests are carried out by defining additional edge “lengths” which incorporate the aggregated coefficients for  $\mathbf{x}$ .

These tests involve additional overhead, of course, but computational tests will show that they can reduce enumeration enough to make this overhead worthwhile. In these tests, “LRE-A” will denote the LRE algorithm with the tests of this section included, and “LRE-PA” will denote the use of both pre-processing from section 4.1 and the aggregated-bounds tests of this section.

### 4.3 Feasibility Problem

When multiple side constraints exist, a feasible solution may not be found while optimizing  $\underline{z}(\boldsymbol{\lambda})$ . Clearly, the problem of finding a feasible path in a multi-constrained CSPP is NP-complete. When this happens, the basic LRE algorithm begins its path-enumeration phase with the weak upper bound  $\bar{z} = (|V| - 1)c_{\max} + \gamma$ , which this may lead to excessive

enumeration. The alternative method described next focuses on finding a feasible solution directly.

If a feasible path exists, one can be identified by selecting an arbitrary side constraint indexed by  $i'$ , and by then solving:

$$\begin{aligned}
 \mathbf{FIP} \quad & \min_{\mathbf{x}} \mathbf{f}_{i'} \mathbf{x} \\
 \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\
 & \mathbf{f}_i \mathbf{x} \leq g_i \quad \forall i \in I \setminus \{i'\} \\
 & \mathbf{x} \geq \mathbf{0} \text{ and integer.}
 \end{aligned}$$

Any feasible solution of **FIP** with objective value no greater than  $g_{i'}$  is feasible for **CSPIP** and hence the corresponding path provides an upper bound for **CSPIP**. We solve **FIP** using the same LRE algorithm as for the original problem with the modification that the algorithm terminates when a solution with objective value no greater than  $g_{i'}$  is found. “LRE-PAF” will denote the LRE algorithm using this “feasibility-checking enhancement” along with the enhancements from the previous two subsections.

**FIP** has only one fewer side constraint compared to the original CSPP. This is a significant reduction only for small  $|I|$  and cannot account for the improvements seen in our testing. The benefit of using **FIP**, in tightly constrained problems, clearly derives from the fact that  $g_{i'}$  tends to be a fairly tight upper bound on the optimal objective value for **FIP**, while the upper bound  $\bar{z} = (|V| - 1)c_{\max} + \gamma$  is weak for the original CSPP.

## 5 Computational Results

This section describes computational experiments with the LRE algorithm applied to problem instances defined on three classes of networks. We carry out all tests on a laptop computer with a 3 GHz Intel Pentium IV processor, 1 gigabyte of RAM, the Microsoft Windows XP Professional operating system, and with programs written and compiled using Microsoft Visual C++ Version 6.0.

We solve instances of CSPP with at most ten side constraints, so repeated bisection

searches in the coordinate directions (Fox and Landi [12], DeWolfe et al. [8]) suffice to maximize  $\underline{z}(\lambda)$  adequately and quickly. We have verified “adequately” by solving the the LP relaxation of a number of instances of CSPP using an interior-point algorithm. Reported solution times verify “quickly.” All versions of LRE employ the shortest-path algorithm of Pape [24] as a subroutine. This algorithm has exponential worst-case complexity, but performs consistently well on the problem classes studied here. For brevity, we use “D&B” when referring to Dumitrescu and Boland [10].

### 5.1 Small-Scale Problem Instances

The first class of test problems consists of the 24 problem instances from Beasley and Christofides [3]. Each instance has either one or ten side constraints and is solved to optimality. Table 1 displays problem and solution statistics.

Column five of the table gives the “initial optimality gap” after optimizing  $\underline{z}(\lambda)$  but before initiating path enumeration, i.e.,  $100\%(\bar{z} - \underline{z}^*)/\underline{z}^*$ . Similarly, column six gives shows the Lagrangian duality gap, i.e., as  $100\%(z^* - \underline{z}^*)/\underline{z}^*$ . For reference, column eight gives run times from Beasley and Christofides. Those computations were performed using FORTRAN on a CDC 7600 computer, and hence, a direct comparison of run times is difficult. D&B also evaluate these problems, but do not report run times. However, D&B do solve most of these problems using only their pre-processing routines (Column nine of Table 1 indicates whether or not the pre-preprocessing routine suffices to solve the instance.) Even though most of these problems have large duality gaps, and may therefore require a significant amount of path enumeration, the table shows that they present no computational challenge to the basic LRE algorithm.

### 5.2 Grid Networks

Grid networks, with the same structure as those studied by D&B comprise the second class of test problems. The test networks, denoted “Grid( $a, b$ ),” derive from a rectangular grid,  $a$  vertices tall and  $b$  vertices wide, with a separate source vertex  $s$  and sink vertex  $t$  external

| Problem |       |       |       | Initial    | Duality    | Run Time      |               | Presolve |
|---------|-------|-------|-------|------------|------------|---------------|---------------|----------|
|         | $ V $ | $ E $ | $ I $ | gap<br>(%) | gap<br>(%) | LRE<br>(sec.) | B&C<br>(sec.) | D&B      |
| BC1     | 100   | 955   | 1     | 60         | 47         | 0.0           | 1.9           | yes      |
| BC2     | 100   | 955   | 1     | 45         | 34         | 0.0           | 0.9           | yes      |
| BC3     | 100   | 959   | 1     | 33         | 33         | 0.0           | 1.9           | yes      |
| BC4     | 100   | 959   | 1     | 0          | 0          | 0.0           | 1.0           | yes      |
| BC5     | 100   | 990   | 10    | 21         | 21         | 0.0           | 4.6           | yes      |
| BC6     | 100   | 990   | 10    | 16         | 16         | 0.0           | 4.6           | yes      |
| BC7     | 100   | 999   | 10    | 142        | 62         | 0.0           | 4.4           | yes      |
| BC8     | 100   | 999   | 10    | $\infty$   | 227        | 0.0           | 6.3           | no       |
| BC9     | 200   | 2,040 | 1     | 18         | 18         | 0.0           | 2.0           | yes      |
| BC10    | 200   | 2,040 | 1     | 0          | 0          | 0.0           | 2.0           | yes      |
| BC11    | 200   | 1,971 | 1     | 0.1        | 0.1        | 0.0           | 4.0           | yes      |
| BC12    | 200   | 1,971 | 1     | 0.1        | 0.1        | 0.0           | 3.9           | yes      |
| BC13    | 200   | 2,080 | 10    | 133        | 100        | 0.1           | 5.2           | yes      |
| BC14    | 200   | 2,080 | 10    | -          | -          | 0.1           | 9.3           | yes      |
| BC15    | 200   | 1,960 | 10    | $\infty$   | 61         | 0.0           | 9.2           | yes      |
| BC16    | 200   | 1,960 | 10    | $\infty$   | 120        | 0.1           | 12.1          | no       |
| BC17    | 500   | 4,858 | 1     | 41         | 34         | 0.0           | 10.6          | yes      |
| BC18    | 500   | 4,858 | 1     | 32         | 25         | 0.0           | 10.5          | yes      |
| BC19    | 500   | 4,978 | 1     | 0.0        | 0.0        | 0.0           | 11.1          | yes      |
| BC20    | 500   | 4,978 | 1     | 0          | 0          | 0.0           | 6.4           | yes      |
| BC21    | 500   | 4,847 | 10    | 33         | 33         | 0.1           | 13.6          | yes      |
| BC22    | 500   | 4,847 | 10    | 25         | 25         | 0.1           | 13.1          | yes      |
| BC23    | 500   | 4,868 | 10    | 22         | 22         | 0.1           | 26.3          | yes      |
| BC24    | 500   | 4,868 | 10    | 36         | 36         | 0.1           | 26.3          | yes      |

Table 1: Problem statistics and run times for the basic LRE algorithm applied to problems from Beasley and Christofides [3]. Run times on our 3 GHz laptop computer exclude problem-generation time and correspond to a 0% optimality tolerance. BC14 is infeasible and the time reported there is for proving infeasibility. Columns five and six report initial optimality gap and duality gap, respectively. An initial gap of  $\infty$  indicates that no feasible solution was found while optimizing  $\underline{z}(\lambda)$ . The second-to-last column lists the run time, on a CDC 7600 computer, reported by Beasley and Christofides. The last column specifies whether or not D&B solve the problem with pre-processing alone.

to the grid. The source  $s$  connects to each vertex in the leftmost column of the grid, and each vertex in the rightmost column connects to  $t$ . Each vertex  $u$  within the grid has (up to) three edges  $(u, v)$  directed out of it, up, down, and from left to right, as long as the vertex  $v$  exists in the grid. Edge lengths and weights are uniform, randomly generated integers in the range  $[1, 10]$  for vertical edges, and in the range  $[80, 100]$  for horizontal edges. For each  $i \in I$ , weight limits are  $g_i = \alpha g_{\max, i} + (1 - \alpha)g_{\min, i}$ , where  $g_{\min, i}$  denotes the total weight of the minimum-weight path with respect to  $i$ , and  $g_{\max, i}$  denotes the total weight, with respect to  $i$ , of the shortest path (with respect to  $c$ ). As in D&B, we examine  $\alpha$  set to the low (L), medium (M), and high (H) values of 0.05, 0.50, and 0.95, respectively: The “L-instances” are tightly constrained, the “H-instances” are loosely constrained, and the “M-instances” are somewhere inbetween.

### 5.2.1 Singly Constrained CSPPs on Small Grid Networks

Table 2 shows the run times for ten singly constrained instances also solved and reported by D&B. The data for these instances (only) were obtained from one of those authors who indicates that, for a given setting of  $a$ ,  $b$  and  $\alpha$ , each represents the most computationally challenging instance extracted from a large set of randomly generated instances (Dumitrescu [9]). The first five are L-instances from “problem class 4-L, Type 2” in D&B; the second five are M-instances from “problem class 4-M, Type 2.”

Table 2, column five, presents the run times from D&B who use a 930 MHz Pentium III computer with 512 megabytes of RAM. These times do not include pre-processing [9]. Using a conversion factor of 0.93/3.0, column six of Table 2 lists estimated run times for their algorithm on our computer. Columns seven and eight of Table 2 report run times for our LRE-A algorithm (which adds the aggregated-bounds tests of section 4.2 to the basic algorithm), using 0% and 5% relative optimality tolerances, respectively. LRE-A uses no pre-processing on these problems, although the data for the problems on Grid(30, 100) were provided to us after pre-processing. The last column of Table 2 shows that LRE-A solves these problems about 95% faster than the D&B algorithm.

| Grid       | Weight Limit | $ V $   | $ E $   | Run Time (sec.) |              |               |               | Speedup (%) |
|------------|--------------|---------|---------|-----------------|--------------|---------------|---------------|-------------|
|            |              |         |         | D&B Tol. 0%     | D&B' Tol. 0% | LRE-A Tol. 0% | LRE-A Tol. 5% |             |
| (30, 100)  | L            | 3,002   | 8,830   | 0.65            | 0.20         | 0.02          | 0.00          | 90.1        |
| (100, 100) | L            | 10,002  | 29,990  | 0.79            | 0.24         | 0.06          | 0.02          | 75.5        |
| (200, 200) | L            | 40,002  | 119,800 | 22.9            | 7.00         | 0.31          | 0.10          | 95.6        |
| (350, 200) | L            | 70,002  | 209,950 | 30.4            | 9.41         | 0.51          | 0.15          | 94.6        |
| (450, 300) | L            | 135,002 | 404,850 | 113.7           | 35.3         | 1.27          | 0.34          | 96.4        |
| (30, 100)  | M            | 3,002   | 8,830   | 0.56            | 0.17         | 0.01          | 0.00          | 94.2        |
| (100, 100) | M            | 10,002  | 29,990  | 3.12            | 0.97         | 0.05          | 0.01          | 94.8        |
| (200, 200) | M            | 40,002  | 119,800 | 21.2            | 6.56         | 0.25          | 0.07          | 96.2        |
| (350, 200) | M            | 70,002  | 209,950 | 36.2            | 11.2         | 0.39          | 0.09          | 96.5        |
| (450, 300) | M            | 135,002 | 404,850 | 144.4           | 44.7         | 1.02          | 0.23          | 97.7        |

Table 2: Comparison of LRE-A to the D&B label-setting algorithm for solving singly constrained CSPPs on grid networks of problem class 4-L, Type 2, and problem class 4-M, Type 2. Each row represents a single problem instance. Times marked “D&B’” are the D&B times scaled by a factor of 0.93/3.0 to account for the slower computer in [10]. “Speedup” is the apparent improvement that LRE-A (0% tolerance) produces over the D&B algorithm. Specifically,  $\text{Speedup} = 100\% \cdot (\text{D\&B}' \text{ sec.} - \text{LRE sec.}) / (\text{D\&B}' \text{ sec.})$ . Results for LRE-A with a 5% tolerance are included for later comparisons.

Table 3 further investigates the behavior of LRE-A by examining the average and standard deviation of run times over 20 randomly generated instances from the problem class in Table 2. All instances solve quickly to optimality (0% tolerance), with the exception of one instance of Grid(450, 300) with medium weight limit. There, the algorithm finds the optimal solution and proves it to be within 0.5% of optimality in 0.4 seconds, but requires 1070 seconds to prove optimality. However, this instance solves to optimality in 61 seconds using the following decomposition technique:

1. Let  $E'$  denote the set of  $a$  edges comprising the “middle echelon” of edges in Grid( $a, b$ ), i.e., the edges between vertex columns  $\lceil b/2 \rceil$  and  $\lceil b/2 \rceil + 1$ . Since  $E'$  essentially defines an  $s$ - $t$  cut with no backward edges, every feasible path from  $s$  to  $t$  must use one and only one edge  $e \in E'$ .
2. For each edge  $e \in E'$ , solve a CSPP that restricts any solution to use that edge. (Simply disallow traversal of  $e' \in E' \setminus \{e\}$ .)
3. The best solution found solves the original CSPP.

Each restricted problem tends to yield a stronger lower bound, which appears to be key to this method’s success. The decomposition also solves faster than might be expected,

| Grid       | Weight Limit | Run Time (sec.)  |      |              |      |
|------------|--------------|------------------|------|--------------|------|
|            |              | Tolerance 0%     |      | Tolerance 5% |      |
|            |              | avg.             | s.d. | avg.         | s.d. |
| (30, 100)  | L            | 0.0              | 0.0  | 0.0          | 0.0  |
| (100, 100) | L            | 0.1              | 0.0  | 0.0          | 0.0  |
| (200, 200) | L            | 0.3              | 0.1  | 0.1          | 0.0  |
| (350, 200) | L            | 0.6              | 0.4  | 0.2          | 0.0  |
| (450, 300) | L            | 1.2              | 0.3  | 0.4          | 0.0  |
| (30, 100)  | M            | 0.0              | 0.0  | 0.0          | 0.0  |
| (100, 100) | M            | 0.1              | 0.0  | 0.0          | 0.0  |
| (200, 200) | M            | 0.5              | 0.8  | 0.1          | 0.0  |
| (350, 200) | M            | 0.5              | 0.2  | 0.1          | 0.0  |
| (450, 300) | M            | 5.0 <sup>†</sup> | 13.0 | 0.1          | 0.0  |
| (30, 100)  | H            | 0.0              | 0.0  | 0.0          | 0.0  |
| (100, 100) | H            | 0.0              | 0.0  | 0.0          | 0.0  |
| (200, 200) | H            | 0.1              | 0.1  | 0.1          | 0.0  |
| (350, 200) | H            | 0.2              | 0.1  | 0.1          | 0.0  |
| (450, 300) | H            | 0.5              | 0.2  | 0.1          | 0.0  |

Table 3: Computational results for the LRE-A algorithm solving singly constrained CSPPs on randomly generated, grid networks. Averages (“avg.”) and standard deviations (“s.d.”) over 20 instances are reported, for optimality tolerances of both 0% and 5%. One instance, in the group indicated by “†,” required application of a special decomposition technique to solve quickly.

because (i) once a feasible solution is found for some edge, the best upper bound  $\bar{z}$  serves to limit the enumeration when considering edges  $e \in E'$  not yet investigated, and (ii) the optimal  $\lambda$  from one restricted CSPP provides an advanced start for optimizing  $z(\lambda)$  in the next problem. A dagger marks the use of this decomposition in Table 3 (only once, for the instance mentioned).

Table 3 indicates that, with one exception, average run times for these problem classes are consistent with the results for the D&B problems in Table 2. And, with one exception, standard deviations are reasonably small. Thus, LRE-A seems to perform well, consistently. Note that the run times with the 5% optimality tolerance are almost negligible.

### 5.2.2 Singly Constrained CSPPs on Large Grid Networks

We now examine the performance of LRE-A on larger grid networks of the same general class tested above. (We also include results for smaller problems here for the sake of comparison.) Table 4 lists average run times, divided by millions of vertices, for LRE-A applied to groups of 20 randomly generated, singly constrained, problem instances of various sizes. We divide

| Grid         | $ V $     | $ E $      | Run-time sec./( $ V /10^6$ ) |     |        |     |      |     |
|--------------|-----------|------------|------------------------------|-----|--------|-----|------|-----|
|              |           |            | Low                          |     | Medium |     | High |     |
|              |           |            | 1%                           | 5%  | 1%     | 5%  | 1%   | 5%  |
| (30, 100)    | 3,002     | 8,830      | 1.5                          | 1.5 | 2.0    | 1.8 | 1.3  | 1.3 |
| (100, 100)   | 10,002    | 29,990     | 3.0                          | 2.5 | 2.0    | 1.9 | 1.5  | 1.5 |
| (200, 200)   | 40,002    | 119,800    | 3.0                          | 2.8 | 2.2    | 2.0 | 1.5  | 1.5 |
| (350, 200)   | 70,002    | 209,950    | 2.9                          | 2.3 | 1.9    | 1.7 | 1.0  | 1.0 |
| (450, 300)   | 135,002   | 404,850    | 3.0                          | 2.6 | 2.1    | 1.9 | 1.0  | 1.0 |
| (600, 400)   | 240,002   | 719,800    | 3.2                          | 2.6 | 2.0    | 1.8 | 1.0  | 1.0 |
| (300, 1000)  | 300,002   | 898,300    | 3.4                          | 2.7 | 2.0    | 1.9 | 1.0  | 1.0 |
| (800, 600)   | 480,002   | 1,439,600  | 7.3                          | 5.6 | 4.4    | 3.7 | 1.5  | 1.5 |
| (1100, 850)  | 935,002   | 2,804,400  | 3.2                          | 2.6 | 2.2    | 1.9 | 1.0  | 1.0 |
| (1000, 1000) | 1,000,002 | 2,999,000  | 3.3                          | 2.6 | 2.1    | 1.9 | 1.0  | 1.0 |
| (1350, 1000) | 1,350,002 | 4,049,350  | 3.5                          | 2.7 | 2.1    | 1.9 | 1.0  | 1.0 |
| (1500, 1100) | 1,650,002 | 4,949,300  | 3.4                          | 2.7 | 2.0    | 1.9 | 1.0  | 1.0 |
| (1650, 1200) | 1,980,002 | 5,939,250  | 3.4                          | 2.7 | 2.0    | 1.9 | 1.1  | 1.1 |
| (1800, 1300) | 2,340,002 | 7,019,200  | 3.5                          | 2.7 | 2.0    | 1.9 | 1.1  | 1.1 |
| (2000, 1500) | 3,000,002 | 8,999,000  | 3.5                          | 2.7 | 2.0    | 1.9 | 1.1  | 1.1 |
| (2200, 1700) | 3,740,002 | 11,218,800 | 3.3                          | 2.8 | 2.3    | 2.0 | 1.1  | 1.1 |
| (2000, 2000) | 4,000,002 | 11,998,000 | 4.2                          | 3.0 | 2.3    | 2.0 | 1.2  | 1.2 |

Table 4: Scaled average run times for LRE-A solving singly constrained CSPPs on large grid networks. (Smaller networks included for comparison.) Averages cover 20 randomly generated instances, and run times per million vertices are listed in the last six columns for low, medium, and high weight limits. The nearly constant values across each column indicate that run times grow only linearly in  $|V|$ . Note that D&B only consider problems of the sizes above the horizontal line.

by this scaling factor to see how run times depend on network size. Note that the longest average run time is only about 16.8 seconds, for Grid(2000, 2000), which has about 4 million vertices and 12 million edges.

Table 4 displays results for low, medium, and high weight limits. Most of these problems are substantially larger than those considered by D&B, who do not consider networks larger than Grid(450, 300). Because these problems are large, we make no attempt to solve them optimally, but, rather, allow 1% and 5% optimality gaps. Standard deviations of the run times (not reported in Table 4) are consistently small, approximately 0.1 seconds per million vertices for low weight limits, and even smaller for the other cases. Results suggest that run times for LRE-A tend to increase only linearly in  $|V|$  for this class of problems.

### 5.2.3 Multiply Constrained CSPPs on Large Grid Networks

Tables 5-7 report results for LRE-PA solving the CSPPs of Table 4, but with two to ten side constraints instead of one. Other than the small problems from Beasley and Christofides [3],

| Grid        | Run Time: Average and Standard Deviation (sec.) |      |         |      |         |      |         |      |          |      |
|-------------|---|------|---------|------|---------|------|---------|------|----------|------|
|             | $ I =2$   |      | $ I =3$ |      | $ I =4$ |      | $ I =5$ |      | $ I =10$ |      |
|             | avg.  | s.d. | avg.    | s.d. | avg.    | s.d. | avg.    | s.d. | avg.     | s.d. |
| (30,100)    | 0.0   | 0.0  | 0.0     | 0.0  | 0.0     | 0.0  | 0.0     | 0.0  | 0.0      | 0.0  |
| (100,100)   | 0.0   | 0.0  | 0.0     | 0.0  | 0.0     | 0.0  | 0.1     | 0.0  | 0.1*     | 0.0  |
| (200,200)   | 0.1   | 0.0  | 0.2     | 0.0  | 0.2     | 0.0  | 0.2     | 0.0  | 0.5      | 0.1  |
| (350,200)   | 0.2   | 0.0  | 0.2     | 0.0  | 0.3     | 0.0  | 0.4     | 0.0  | 1.7      | 0.2  |
| (450,300)   | 0.4   | 0.0  | 0.5     | 0.0  | 0.6     | 0.0  | 0.8     | 0.1  | 2.3      | 0.4  |
| (600,400)   | 0.8   | 0.1  | 1.1     | 0.1  | 1.9     | 0.3  | 2.3     | 0.3  | 8.3      | 1.1  |
| (300,1000)  | 0.8   | 0.0  | 1.0     | 0.1  | 1.3     | 0.1  | 1.7     | 0.1  | 4.9      | 0.5  |
| (800,600)   | 2.3   | 0.3  | 4.2     | 0.7  | 8.1     | 1.6  | 10.4    | 2.2  | 32.5     | 3.9  |
| (1100,850)  | 2.4   | 0.1  | 3.3     | 0.3  | 4.2     | 0.3  | 5.1     | 0.4  | 11.8     | 0.8  |
| (1000,1000) | 2.6   | 0.1  | 3.6     | 0.3  | 4.5     | 0.3  | 5.5     | 0.3  | 12.5     | 0.9  |
| (1350,1000) | 3.6   | 0.2  | 4.9     | 0.3  | 6.4     | 0.4  | 7.5     | 0.6  | 36.8     | 13.0 |
| (1500,1100) | 4.4   | 0.2  | 6.0     | 0.4  | 7.7     | 0.4  | 9.2     | 0.7  | -        | -    |
| (1650,1200) | 5.4   | 0.2  | 7.2     | 0.5  | 9.3     | 0.6  | 11.8    | 0.5  | -        | -    |
| (1800,1300) | 6.2   | 0.3  | 8.8     | 0.6  | 28.6    | 8.0  | -       | -    | -        | -    |
| (2000,1500) | 9.0   | 2.9  | -       | -    | -       | -    | -       | -    | -        | -    |

Table 5: Solution statistics for LRE-PA solving multiply-constrained CSPPs on large grid networks. The  $|I|$  side constraints have high (“H”) weight limits, and the optimality tolerance is 5%. Averages cover 20 randomly generated instances. Pre-processing has no effect on these instances and all are feasible. Results for a 1% tolerance gives nearly identical results except for the problem marked with an asterisk. One instance of this problem solves slowly at this lower tolerance, but the decomposition technique solves it in 93.7 seconds. We do not solve problems marked with “-” because of computer memory limitations.

D&B do not solve any multiply constrained CSPPs, so we can make no comparisons with their algorithm here.

Table 5 lists solution statistics for problems with the high (“H”) weight limit and a 5% optimality gap. Because these problems are loosely constrained, pre-processing removes no edges (although the overhead is minimal). Using a 1% tolerance (results not shown) gives nearly identical solution statistics with one exception indicated by an asterisk (group Grid(100,100) with  $|I|=10$ ). One instance in this group of 20 did not solve within 30 minutes, but it solves in 93.7 seconds using decomposition. When including the decomposition run-time, we obtain an average time of 5.3 seconds for this group.

Next, we briefly consider LRE-PA applied to the large-grid CSPPs but with low (“L”) weight limits. We omit the corresponding table because: (i) All 1340 randomly generated instances are infeasible, (ii) the pre-processing routine proves this by, essentially, removing all edges in each network, and (iii) average run times are all smaller than the corresponding times in Table 5.

The computational evidence provided by the “H” and “L” problems indicates that LRE-PA can easily solve certain large, loosely constrained or over-constrained CSPPs, even when the problems have a moderate number of side constraints: Pre-processing tends to identify infeasibility quickly for the over-constrained problems, while the loosely constrained problems have small duality gaps, which the algorithm closes to sufficient accuracy quickly.

We next consider the more challenging problems of the “M” variety.

Tables 6 and 7 present solution statistics on for the large-grid CSPPs with medium weight limits, using 1% and 5% optimality tolerances, respectively. Parenthetical superscripts in these tables indicate how many instances (out of 20) did not solve in 30 minutes. Each of these does solve, however, when the decomposition is applied, and the statistics for these problems have been computed using the decomposition-enabled times.

Results in Table 6 indicate that, with only three exceptions, LRE-PA quickly solves the 580 M-instances with two or three side constraints to within 1% of optimality. (All instances are feasible.) The decomposition technique solves the exceptions reasonably quickly.

Instances with ten side constraints also solve efficiently at 1%. All these instances are infeasible, but not as “clearly” as in the L-instances. That is, pre-processing does not remove all edges in all instances for these problems.

The problems with four and five side constraints include several “barely feasible” or “barely infeasible” instances. Hence, we cannot expect that the process of optimizing  $\underline{z}(\lambda)$  will always generate a feasible solution and a good upper bound. But, enumeration using the weak upper bound  $(|V| - 1)c_{\max} + \gamma$  can be extremely slow for moderately large networks. In view of this, we equip LRE-PA with the feasibility-checking enhancement of section 4.3, to create LRE-PAF. Solution statistics in Table 6 reflect application of this enhanced algorithm.

The combination of several side constraints, many nearly feasible paths, and moderate or large network size can increase run times substantially when using this tight optimality tolerance. Table 6 shows that smaller instances with four and five side constraints can be solved, possibly requiring decomposition, but some larger instances may simply become too difficult. We note that many of the instances with four and five side constraints do solve

| Grid        | Average Run Time and Standard Deviation (sec.) |      |                     |      |                     |      |         |      |          |      |
|-------------|--|------|---------------------|------|---------------------|------|---------|------|----------|------|
|             | $ I =2$  |      | $ I =3$             |      | $ I =4$             |      | $ I =5$ |      | $ I =10$ |      |
|             | avg.   | s.d. | avg.                | s.d. | avg.                | s.d. | avg.    | s.d. | avg.     | s.d. |
| (30,100)    | 0.0  | 0.0  | 0.0                 | 0.1  | 8.8                 | 31.1 | 142     | 376  | 2.1      | 6.2  |
| (100,100)   | 0.1  | 0.0  | 1.5 <sup>(1)</sup>  | 3.0  | 13.4 <sup>(1)</sup> | 29.2 | 24.0    | 70.3 | 2.4      | 7.0  |
| (200,200)   | 0.2  | 0.1  | 3.0                 | 8.3  | 258 <sup>(4)</sup>  | 423  | [19]    |      | 0.3      | 0.0  |
| (350,200)   | 0.4  | 0.2  | 2.6                 | 3.3  | 161 <sup>(2)</sup>  | 280  | [16]    |      | 22.5     | 95.6 |
| (450,300)   | 0.8  | 0.4  | 0.8                 | 0.0  | [18]                |      | [11]    |      | 45.8     | 194  |
| (600,400)   | 2.5  | 1.2  | 64.8                | 247  | [16]                |      | [6]     |      | 3.2      | 0.0  |
| (300,1000)  | 1.5  | 0.4  | 9.6 <sup>(1)</sup>  | 23.9 | 12.6                | 9.3  | [17]    |      | 2.8      | 0.0  |
| (800,600)   | 10.5   | 5.2  | 57.8                | 17.8 | [19]                |      | [12]    |      | 10.8     | 0.1  |
| (1100,850)  | 4.7  | 1.1  | 79.0 <sup>(1)</sup> | 286  | [19]                |      | [15]    |      | 7.5      | 0.0  |
| (1000,1000) | 4.9  | 0.5  | 14.7                | 5.3  | [18]                |      | [11]    |      | 8.0      | 0.0  |
| (1350,1000) | 6.9  | 1.0  | 20.0                | 7.8  | 60.8                | 65.2 | [15]    |      | 19.7     | 9.0  |
| (1500,1100) | 8.5  | 0.9  | 26.0                | 7.9  | 58.3                | 49.9 | [16]    |      | -        | -    |
| (1650,1200) | 10.6   | 3.0  | 41.7                | 37.2 | [18]                |      | [16]    |      | -        | -    |
| (1800,1300) | 12.0   | 1.6  | 36.3                | 9.7  | 71.9                | 44.1 | -       | -    | -        | -    |
| (2000,1500) | 17.0   | 4.8  | -                   | -    | -                   | -    | -       | -    | -        | -    |

Table 6: Solution statistics for LRE-PAF, with a 1% optimality tolerance, applied to large-grid CSPPs with medium (“M”) weight limits and  $|I|$  weights. The parenthetic superscripts specify the number of instances (out of 20) that did not solve in 30 minutes. These instances do solve using decomposition, and those decomposition times are used to compute solution statistics. For problems where LRE-PAF did not solve all 20 instances in 30 minutes, with or without decomposition, we report the number that did in brackets. We did not attempt to solve problems marked with “-” because of computer-memory limitations.

quickly, but when one or more of the 20 instances in a group does not solve in 30 minutes by means of LRE-PAF, with or without decomposition, we place the number of solved instances in brackets.

Table 7 reports solution statistics for LRE-PAF applied to the problems of Table 6, but with a 5% optimality tolerance. Run times decrease significantly in some cases. In particular, all instances with four side constraints now solve easily using LRE-PAF, and only a few large instances with five side constraints remain unsolved after 30 minutes. Note that some of the standard deviations in Tables 6 and 7 are large, which indicates substantial variation in problem difficulty with some of these problem sets.

| Grid        | Average Run Time and Standard Deviation (sec.) |      |         |      |         |      |                     |      |          |      |
|-------------|--|------|---------|------|---------|------|---------------------|------|----------|------|
|             | $ I =2$  |      | $ I =3$ |      | $ I =4$ |      | $ I =5$             |      | $ I =10$ |      |
|             | avg.   | s.d. | avg.    | s.d. | avg.    | s.d. | avg.                | s.d. | avg.     | s.d. |
| (30,100)    | 0.0  | 0.0  | 0.0     | 0.0  | 0.1     | 0.2  | 1.1                 | 2.1  | 2.1      | 6.2  |
| (100,100)   | 0.0  | 0.0  | 0.1     | 0.1  | 1.0     | 1.7  | 1.8                 | 2.0  | 2.4      | 7.0  |
| (200,200)   | 0.2  | 0.0  | 0.2     | 0.1  | 2.1     | 3.1  | 47.1 <sup>(2)</sup> | 102  | 0.3      | 0.0  |
| (350,200)   | 0.2  | 0.0  | 0.3     | 0.0  | 3.5     | 7.1  | 100 <sup>(2)</sup>  | 206  | 22.5     | 95.6 |
| (450,300)   | 0.5  | 0.0  | 0.8     | 0.0  | 38.6    | 156  | 39.9 <sup>(2)</sup> | 63.5 | 45.8     | 194  |
| (600,400)   | 1.5  | 0.0  | 2.3     | 0.2  | 9.5     | 21.3 | [19]                |      | 3.2      | 0.0  |
| (300,1000)  | 1.3  | 0.0  | 1.8     | 0.3  | 2.3     | 0.1  | 12.5                | 24.0 | 2.8      | 0.0  |
| (800,600)   | 5.5  | 0.2  | 11.9    | 0.5  | 21.5    | 3.6  | [18]                |      | 10.8     | 0.1  |
| (1100,850)  | 3.7  | 0.0  | 5.1     | 0.2  | 6.7     | 0.3  | 54.7                | 91.6 | 7.5      | 0.0  |
| (1000,1000) | 4.0  | 0.0  | 5.4     | 0.2  | 7.1     | 0.4  | [19]                |      | 8.0      | 0.0  |
| (1350,1000) | 5.5  | 0.1  | 7.5     | 0.2  | 9.7     | 0.5  | [19]                |      | 19.7     | 9.0  |
| (1500,1100) | 6.7  | 0.0  | 9.2     | 0.2  | 11.9    | 0.5  | 39.5                | 106  | -        | -    |
| (1650,1200) | 8.3  | 0.0  | 11.4    | 0.2  | 14.9    | 0.7  | [19]                |      | -        | -    |
| (1800,1300) | 9.7  | 0.0  | 13.4    | 0.3  | 36.2    | 10.6 | -                   | -    | -        | -    |
| (2000,1500) | 12.8   | 1.8  | -       | -    | -       | -    | -                   | -    | -        | -    |

Table 7: Solution statistics for LRE-PAF, with a 5% optimality tolerance, applied to large-grid CSPPs with medium (“M”) weight limits and  $|I|$  weights. See the caption of Table 6 for additional details.

### 5.3 Routing Military Units through a Road Network

Our third class of test problems derives from planning the movement of a military unit through a road network. Consider a small convoy, which must move from junction  $s$  in the network to junction  $t$ , in a limited amount of time. Planners wish to select a route that meets the time limit, but minimizes “risk.” We can formulate this problem as a CSPP with one side constraint.

Let weight  $f_e = f_{1e}$  of edge  $e = (u, v)$  represent the time required to traverse road segment  $e$ , and let length  $c_e$  represent the “risk” of traversing  $e$ . The convoy will travel with civilian traffic and obey speed limits, so  $f_e$  equals the physical length of  $e$  divided by its speed limit. Planners deem that larger roads with higher speed limits are riskier, increasing the unit’s exposure to mines, ambushes, etc. Hence, as a surrogate for risk we set  $c_e = f_e \beta_e$ , where  $\beta_e = 5.0, 2.0, 1.0, 0.5$ , and  $0.2$  when  $e$  is a major highway, a minor highway, a major expressway, a minor expressway, or a local road, respectively. Consequently, the optimal route traverses small, slow roads as much as possible, given the time limit.

Tables 8 and 9 present computational results, using 1% and 5% relative optimality tolerance, respectively, for the LRE algorithm, with and without various enhancements, applied

| $g$  | Edges pre-<br>proc. (%) | Initial gap (%) |      | Duality Gap (%) |      | Run Time (sec.) |       |       |        |
|------|-------------------------|-----------------|------|-----------------|------|-----------------|-------|-------|--------|
|      |                         | No pre.         | Pre. | No pre.         | Pre. | LRE             | LRE-P | LRE-A | LRE-PA |
| 240* | 100                     | –               | –    | –               | –    | 0.0             | 0.0   | 0.0   | 0.0    |
| 250  | 92                      | 2.1             | <1.0 | 2.1             | <1.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 260  | 83                      | 1.7             | 1.7  | <1.0            | <1.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 270  | 77                      | 16.6            | 16.6 | <1.0            | <1.0 | 36.1            | 38.0  | 7.0   | 7.1    |
| 280  | 73                      | 15.0            | 14.2 | 6.2             | 5.2  | 3.6             | 2.1   | 1.0   | 0.5    |
| 290  | 70                      | 35.3            | 11.7 | 6.5             | 3.8  | 1.3             | 0.1   | 0.4   | 0.0    |
| 300  | 68                      | 64.4            | 33.2 | 9.2             | 3.3  | 885.9           | 67.7  | 217.5 | 16.5   |
| 310  | 64                      | <1.0            | <1.0 | <1.0            | <1.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 320  | 59                      | 12.9            | 12.9 | 1.9             | 1.9  | 0.2             | 0.2   | 0.1   | 0.1    |
| 330  | 49                      | <1.0            | <1.0 | <1.0            | <1.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 340  | 45                      | 1.1             | 1.1  | <1.0            | <1.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 350  | 40                      | <1.0            | <1.0 | <1.0            | <1.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 360  | 37                      | 1.2             | 1.2  | <1.0            | <1.0 | 0.0             | 0.0   | 0.0   | 0.0    |

Table 8: Computational results for planning the movement of a military convoy through a road network. The relative optimality tolerance is 1%. We report edges removed by pre-processing, initial gap (i.e., the relative gap between  $\bar{z}$  and  $\underline{z}^*$ ), duality gap (i.e., the relative gap between  $z^*$  and  $\underline{z}^*$ ), and run times for various algorithmic variants. The algorithms using the enhancements of section 4 (LRE-P, LRE-A, LRE-PA) significantly outperform the basic LRE algorithm in many instances. The instance marked with an asterisk is infeasible.

to the road network in Maryland, Virginia, and Washington, D.C. [6]. The data include all interstate, state, and county roads with speed limits over 25 miles per hour, and generate a graph with 3,670 vertices and 9,876 edges. Road segments with speed limits 65, 55, 50, 45, and 30 miles per hour are categorized as major highways, minor highways, and so on, respectively. Tables 8 and 9 displays computational results for a range of hypothesized time limits in minutes. Note that it is impossible to reach location  $t$  in less than 240 minutes, and no improvement in risk accrues beyond 360 minutes.

The second column of Tables 8 and 9 display the percentage of edges that can be removed by pre-processing. Most edges are eliminated by pre-processing for tightly constrained problems, but, even a modest amount of pre-processing can tighten the Lagrangian lower bound substantially. Columns 3-6 in Tables 8 and 9 exemplify this. For the case with  $g = 300$ , the tightening in the Lagrangian lower bound reduces run time significantly, as seen from comparing the basic LRE algorithm (column seven) with its enhanced version including pre-processing (column eight). We observe a similar reduction when comparing LRE-A (column nine) with LRE-PA (column ten), which adds pre-processing. This effect is evident for both optimality tolerances.

| $g$  | Edges pre-proc. (%) | Initial gap (%) |      | Duality Gap (%) |      | Run Time (sec.) |       |       |        |
|------|---------------------|-----------------|------|-----------------|------|-----------------|-------|-------|--------|
|      |                     | No pre.         | Pre. | No pre.         | Pre. | LRE             | LRE-P | LRE-A | LRE-PA |
| 240* | 100                 | –               | –    | –               | –    | 0.0             | 0.0   | 0.0   | 0.0    |
| 250  | 92                  | <5.0            | <5.0 | <5.0            | <5.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 260  | 83                  | <5.0            | <5.0 | <5.0            | <5.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 270  | 77                  | 16.6            | 16.6 | <5.0            | <5.0 | 0.4             | 0.4   | 0.1   | 0.1    |
| 280  | 73                  | 14.9            | 14.2 | 6.2             | 5.2  | 0.3             | 0.1   | 0.1   | 0.1    |
| 290  | 70                  | 35.3            | 11.7 | 6.5             | <5.0 | 0.1             | 0.0   | 0.0   | 0.0    |
| 300  | 68                  | 64.4            | 33.2 | 9.2             | <5.0 | 14.3            | 1.1   | 4.5   | 0.3    |
| 310  | 64                  | <5.0            | <5.0 | <5.0            | <5.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 320  | 59                  | 12.9            | 12.9 | <5.0            | <5.0 | 0.1             | 0.1   | 0.0   | 0.0    |
| 330  | 49                  | <5.0            | <5.0 | <5.0            | <5.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 340  | 45                  | <5.0            | <5.0 | <5.0            | <5.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 350  | 40                  | <5.0            | <5.0 | <5.0            | <5.0 | 0.0             | 0.0   | 0.0   | 0.0    |
| 360  | 37                  | <5.0            | <5.0 | <5.0            | <5.0 | 0.0             | 0.0   | 0.0   | 0.0    |

Table 9: Computational results for planning the movement of a military convoy through a road network. The relative optimality tolerance is 5%. See the caption of Table 8 for additional details.

A similar comparison illustrates the effect of the aggregated-bounds tests. For the cases with time limits  $g = 270$  and  $g = 300$ , these tests reduces run times significantly; compare columns seven and nine. We observe a similar reduction when comparing the LRE algorithm with pre-processing (column eight) with the “full algorithm,” which includes both pre-processing and aggregated-bounds tests; compare columns eight and ten. Overall, Tables 8 and 9 indicate that the LRE algorithm, particularly with enhancements, does solve the CSPP on this real-world network efficiently.

## 6 Conclusions

We have described an improved algorithm for solving the constrained shortest-path problem (CSPP). Our “LRE algorithm” Lagrangianizes all side constraints, optimizes the resulting Lagrangian function, defines new edge lengths through the Lagrangian function, and enumerates all near-shortest paths in order to close any remaining duality gap.

Testing on a variety of instances with up to ten side constraints indicates that the LRE algorithm can be significantly faster than the most recently published label-setting algorithm, and can handle instances with millions of vertices and edges.

The basic LRE algorithm solves many large problems quickly, but several enhancements can sometimes improve performance:

1. Pre-processing to eliminate edges that can be determined, *a priori*, not to lie on any optimal path,
2. Adding aggregated constraints to help exclude some infeasible paths from being followed in the path-enumeration phase,
3. Solving an initial feasibility problem, which simply represents a variant of the original CSPP, when the process of optimizing the Lagrangian function does not yield a feasible solution, and
4. Decomposing the problem into a series of restricted versions, in which every feasible path must traverse a specific edge.

Versions of the first three enhancements, even when not required, do not substantially add to computational overhead, so we recommend them as standard additions to the basic algorithm.

## Acknowledgments

The authors thank the Air Force Office of Scientific Research, the Office of Naval Research and the Naval Postgraduate School sabbatical program for funding this research. Kevin Wood also thanks the University of Auckland, Department of Engineering Science, for providing support in the preparation of this paper.

## References

- [1] Ahuja, R.K., Magnanti, T.L., and J.B. Orlin, (1993), *Network Flows*, Prentice Hall, Englewood Cliffs, New Jersey.
- [2] Aneja, Y., Aggarwal, V., and Nair, K., (1983), "Shortest Chain Subject to Side Conditions," *Networks*, 13, pp. 295–302.
- [3] Beasley, J. and Christofides, N., (1989), "An Algorithm for the Resource Constrained Shortest Path Problem," *Networks*, 19, pp. 379–394.
- [4] Boerman, D., (1994), "Finding an Optimal Path Through a Mapped Minefield," Master's Thesis, Naval Postgraduate School, Monterey, California, March.

- [5] Byers, T.H. and Waterman, M.S., (1984), "Determining Optimal and Near-Optimal Solutions When Solving Shortest Path Problems by Dynamic Programming," *Operations Research*, 32, pp. 1381-1384.
- [6] Carlyle, W.M. and Wood, R.K., (2003), "Near-Shortest and K-Shortest Simple Paths," *Networks*, to appear.
- [7] Day, P.R. and Ryan, D.M., (1997), "Flight Attendant Rostering for Short-Haul Airline Operations," *Operations Research*, 45, pp. 649-661.
- [8] DeWolfe, D., Stevens, J. and Wood, K., (1993), "Setting Military Reenlistment Bonuses," *Naval Research Logistics*, 40, pp. 143-160.
- [9] Dumitrescu, I., (2005), Private communication, 20 April 2005.
- [10] Dumitrescu, I. and Boland, N., (2003), "Improved Preprocessing, Labeling and Scaling Algorithm for the Weight-Constrained Shortest Path Problem," *Networks*, 42, pp. 135-153.
- [11] Fisher, M.L., (1981), "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science*, 27, pp. 1-18.
- [12] Fox, B.L. and Landi, D.M., (1970), "Searching for the Multiplier in One-Constraint Optimization Problems," *Operations Research*, 18, pp. 253-262.
- [13] Gamache, M., Soumis, F., Marquis, M. and Desrosiers, J., (1999), "A Column Generation Approach for Large-Scale Aircrew Rostering Problems," *Operations Research*, 47, pp. 247-263.
- [14] Garey, M.R. and Johnson, D.S., (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco.
- [15] Hadjiconstantinou, E. and Christofides, N., (1999), "An Efficient Implementation of an Algorithm for Finding  $K$  Shortest Simple Paths," *Networks*, 34, pp. 88-101.
- [16] Handler, G. and Zang, I., (1980), "A Dual Algorithm for the Constrained Shortest Path Problem," *Networks*, 10, pp. 293-310.
- [17] Joks, H., (1966), "The Shortest Route Problem with Constraints," *Journal of Mathematical Analysis and Application*, 14, pp. 191-197.
- [18] Kaufman, D.E. and Smith, R.L., (1993), "Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Applications," *IVHS Journal*, 1, pp. 1-11.
- [19] Korkmaz, T. and Krunz, M., (2001), "Multi-Constrained Optimal Path Selection," in *Proceedings of the IEEE INFOCOM 2001 Conference, Vol. 2*, Anchorage, Alaska, April 22-26, pp. 834-843.
- [20] Latourelle, J.L., Wallet, B.C. and Copeland, B., (1998), "Genetic Algorithm to Solve Constrained Routing Problems with Applications for Cruise Missile Routing," in *Applications and Science of Computational Intelligence, Proceedings of SPIE, Vol. 3390*, Society of Photo-optical Instrumentation Engineers, Bellingham, Washington, USA, pp. 490-500.

- [21] Lee, S.H.K., (1995), "Route Optimization Model for Strike Aircraft," Master's Thesis, Naval Postgraduate School, Monterey, California, September.
- [22] Nachtigall, K., (1995), "Time Depending Shortest-path Problems with Applications To Railway Networks," *European Journal of Operational Research*, 83, pp. 154–166.
- [23] Nygaard, R., Melnikov, G. and Katsaggelos, A.K., (2001), "A Rate Distortion Optimal ECG Coding Algorithm," *IEEE Transactions on Biomedical Engineering*, 48, pp. 28–40.
- [24] Pape, U., (1974), "Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem," *Mathematical Programming*, 7, pp. 212–222.
- [25] Vance, P.H., Barnhart, C., Johnson, E.L. and Nemhauser, G.L., (1997), "Airline Crew Scheduling: A New Formulation and Decomposition Algorithm," *Operations Research*, 45, pp. 188–200.
- [26] Russell, S. and Norvig, P., (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall.
- [27] Wolsey, L.A., (1998), *Integer Programming*, John Wiley & Sons, New York.
- [28] Zabrankin, M., Uryasev, S. and Pardalos, P., (2001), "Optimal Risk Path Algorithms" in *Cooperative Control and Optimization*, R. Murphey and P. Pardalos, eds., Kluwer Academic Publishers, Dordrecht, pp. 273–299.

## Appendix

### Path-Enumeration Subroutine for LRE Algorithm to Solve CSPP

INPUT: A directed graph  $G = (V, E)$  in adjacency-list format,  $s, t$ , edge length

vector  $\mathbf{c} \geq \mathbf{0}$ , side-constraint data for  $F\mathbf{x} \leq \mathbf{g}$  with  $\mathbf{f}_i \geq \mathbf{0}$ ,

optimal or near-optimal Lagrangian vector  $\boldsymbol{\lambda}$  for **CSPLR**,

upper bound  $\bar{z}$ , lower bound  $\underline{z}(\boldsymbol{\lambda})$  and optimality tolerance  $\delta \geq 0$ .

OUTPUT: A  $\delta$ -optimal shortest path  $\mathbf{x}^*$  satisfying  $F\mathbf{x}^* \leq \mathbf{g}$ , if such a path exists.

NOTE: “firstEdge( $v$ )” points to the beginning of a linked list of edges directed out of  $v$

```

{
   $\mathbf{c}' \leftarrow \mathbf{c} + \boldsymbol{\lambda}F$ ;
  /* Add a “0-th side constraint” to limit enumeration based on  $\mathbf{c}'$  */
   $I^+ \leftarrow I \cup \{0\}$ ;  $\mathbf{f}_0 \leftarrow \mathbf{c}$ ;  $g_0 \leftarrow \bar{z}$ ;
  /* The following requires just one backwards shortest-path calculation */
  for ( all  $v \in V$  ) {  $d(v) \leftarrow$  minimum distance, in terms of  $\mathbf{c}'$ , from  $v$  to  $t$ ; }
  for ( each side constraint  $i \in I^+$  ) {
    /* Solve a backwards shortest-path problem using edge “lengths”  $\mathbf{f}_i$  */
    for ( all  $v \in V$  ) {  $d_i(v) \leftarrow$  minimum weight, in terms of  $\mathbf{f}_i$ , from  $v$  to  $t$ ; }
  }
  for( all  $v \in V$  ) { nextEdge( $v$ )  $\leftarrow$  firstEdge( $v$ ); }
   $L(s) \leftarrow -\boldsymbol{\lambda}\mathbf{g}$ ; /* Initialize path length with the Lagrangian constant term */
  for( all  $i \in I^+$  ) {  $L_i(s) \leftarrow 0$ ; } /* Initial path weight with respect  $\mathbf{f}_i$  is 0 */
  theStack  $\leftarrow s$ ; onStack( $s$ )  $\leftarrow$  true; onStack( $v$ )  $\leftarrow$  false  $\forall v \in V \setminus \{s\}$ ;
  while ( theStack is not empty ) {
     $u \leftarrow$  vertex at the top of theStack;
    if ( nextEdge( $u$ )  $\neq \emptyset$  ) {
       $e \leftarrow$  the edge pointed to by nextEdge( $u$ ); /*  $e = (u, v)$  */
      increment nextEdge( $u$ );
      if ( (onStack( $v$ ) = false) and ( $L(u) + c'_e + d(v) < \bar{z} - \delta$ )
        and ( $L_i(u) + f_{ie} + d_i(v) \leq g_i \forall i \in I^+$ ) ) {
        if (  $v = t$  ) { /* An improved solution has been found */
          Represent the feasible path encoded as theStack  $\cup \{t\}$  through
          its edge-incidence vector  $\hat{\mathbf{x}}$ ;
           $\bar{z} \leftarrow \mathbf{c}\hat{\mathbf{x}}$ ;  $g_0 \leftarrow \bar{z}$ ;  $\mathbf{x}^* \leftarrow \hat{\mathbf{x}}$ ;
          /* Preemptive termination is possible in the following step */
          if (  $\bar{z} - \underline{z}(\boldsymbol{\lambda}) \leq \delta$  ) goto Finish;
        } else {
          push  $v$  on theStack; onStack( $v$ )  $\leftarrow$  true;
           $L(v) \leftarrow L(u) + c_e$ ;
          for ( all  $i \in I^+$  ) {  $L_i(v) \leftarrow L_i(u) + f_{ie}$ ; }
        }
      }
    }
    } else {
      Pop  $u$  from theStack; onStack( $u$ )  $\leftarrow$  false;
      nextEdge( $u$ )  $\leftarrow$  firstEdge( $u$ );
    }
  }
}
Finish: If  $\mathbf{x}^*$  is empty Print ( Problem is infeasible ), otherwise Print (  $\mathbf{x}^*$  );
}
```